USING REPAST AND JAVA TO ENHANCE THE NOM PROJECT: DOCUMENTATION

SUMMER 2003

DR. MADEY
RYAN KENNEDY
UNIVERSITY OF NOTRE DAME

MAJOR ACCOMPLISHMENTS

Creating the natural organic matter (NOM) graphical user interface (GUI) was my major accomplishment for the summer. My successful completion of the project was possible because I had adequately taught myself Java, RePast, and some Swarm. I also contributed to the testing of the NOM Simulator by performing load balancing tests and restructuring some of the Swarm code so that its conversion into RePast was much simpler. Overall, I had a fun and educational summer.

REFERENCES AND TOOLS

Cabaniss, Steve. *White Papers: Modeling and Stochastic Simulation of NOM Reactions*. University of Notre Dame, 2002.

Deitel & Deitel. *Java: How to Program (Third Edition)*. Upper Saddle River, NJ: Prentice Hall, 1999.

Huang, Yingping. *Infrastructure, Query Optimization, Data Warehousing and Data Mining for Scientific Simulation*: Master's Thesis. University of Notre Dame, 2002.

Madey, Gregory R., Ph.D..

Xiang, Xiaorong. *Agent-Based Scientific Applications and Collaboration Using Java*: Master's Thesis. University of Notre Dame, 2003.

Java Version 1.4.1
RePast Version 2.0.1
Julio Dovalina (REU Student)
http://java.sun.com/j2se/1.4.1/docs/api/
http://repast.sourceforge.net

LOG OF ACTIVITIES

WEEK 1

I began my work this summer by reading Yingping's thesis and then Xiaorong's thesis. This took me about 2 days. Next, I began my study of the aforementioned Deitel and Deitel Java book. Having no prior coding experience in Java, I carefully read chapters 1-9, 18-19, and 25. These chapters familiarized me with a general overview of Java and also some detail on JDBC, Servlets, and JavaBeans. My study of chapters 1-9 proved most useful during my work this summer. Along the way, I occasionally would write test programs of my own to further increase my grasp of the material. Reading and studying the Java book took the remainder of the first week I was here.

WEEK 2

I began my second week by restructuring and cleaning some of the Swarm core simulation code as instructed by Xiaorong. She told me this would be beneficial when the code was converted into RePast.

Specifically, I split the *move* method into a *move* and *sorption* method and moved each of them and the *react* method from *ModelSwarm.java* to *Molecule.java*. This took me 2-3 days, as I was still becoming familiar with the Java and Swarm environments and as well as the NOM project. During this time (and for the remainder of the summer), I frequently visited the Java API website for reference.

Later in the week, I worked with Yingping to discuss some preliminary methods for testing his load balancing mechanism. I also worked with Julio to come up with suggestions regarding how to improve the NOM site. We created with a list of suggestions and submitted it to Xiaorong and Yingping.

WEEK 3

I finished restructuring the core simulation code this week and then tested it for memory leaks and verified it produced the correct results. Xiaorong helped me with this and fortunately the code passed both tests. After the testing, I read the White Papers, which gave me a good background on the chemistry and idea behind the NOM simulator.

My plans for the remainder of the summer were also discussed this week. We decided that I would concentrate the remainder of my work on creating the NOM GUI.

WEEK 4

I began my fourth week by installing RePast on my allocated space on Xiaorong's machine, tenor. After it was successfully installed, I explored the demo simulations and source code. I found it valuable to edit their provided examples to test new variations of the code. Using this technique helped most when learning RePast. Next, I read the following How-To Documents from the RePast website:

- How To Run a RePast Simulation
- How To use the GUI
- How to Build a RePast Model - 1
- How to Build A RePast Model - 2
- How to Use a Schedule
- How to use spaces
- How To Create Displays
- How to Work with Random Numbers
- How To Create Charts

These documents were very helpful and got me excited about my work in RePast. I also read the FAQ section and familiarized myself with the API Documentation.

In addition to learning about RePast, I continued to edit the NOM code and was able to get it to compile using the *javac* (Java)compiler instead of the *javacswarm* (JavaSwarm) compiler. The only thing left to covert was the use of Swarm's random number generator, which I had commented out for testing. This meant that the code was no longer dependent on Swarm.

WEEK 5

I began my fifth week by converting to the RePast random number generator, utilizing RePast's *getNextDoubleFromTo()* method. I also removed most of the other leftover unused Swarm code and ran the simulation for 4000 timesteps to verify that it was working properly and that there were still no memory leaks. Fortunately, it passed the tests. The simulation was now able to compile successfully using the *javac* compiler and able to run using the syntax *java StartMolecule 102 batch* instead of *javaswarm StartMolecule 102 batch*, where 102 represents the simulation number.

Next, I was able to begin the GUI. I studied the *Life* and *HeatBugs* source codes to help me get started. After a lot of experimenting, I was able to get a primitive display of the molecules.

WEEK 6

I continued my work on the GUI this week. I added a color-coded legend that displays the name of each type of molecule in the simulation and its molecular weight. The code references data read from the simulation that is invoked to get the number of molecule types and generates the displayed legend from that. I also added a graph (that the user can enable or disable) that plots the Molecule Type vs. Number of Molecules Adsorbed. The plot updates at the user-specified interval. The settings window displays some user-controllable variables and the molecules in the display are probeable, meaning you can display their unique characteristics.

I also performed the load balancing tests this week. I began by choosing the parameters for the simulations. I chose to run 10 simulations that would be easy on the machine and 6 that would be stressful (meaning they would take up much more of the CPU). I submitted 4 of the easy simulations, then 4 of the stressful ones, then 4 of the easy ones, then 1 stressful one, 2 easy ones, and finally 1 stressful simulation. The mechanism always started the simulation on the machine with the lowest load average; therefore, the test was successful. Load average is defined as the number of processes waiting to be executed in the waiting queue over time and was accessible through the Simulation Manager webpage.

WEEK 7

Following a meeting with some of the scientists also working on the project, I made some of the changes they requested. Specifically, I made it so the molecules were colored according to a gradient in which the molecules with the highest molecular weight were red and the lowest were blue. This color-coding is dependent on the molecules being entered into the simulation data in ascending order of their molecular weight. I also added a new dataset that plotted the Molecular Weight vs. Number of

Desorbed Molecules on the same graph as the other plot. Finally, I made it so that the GUI supports up to 256 different molecule types and colors.

DIFFICULTIES

The first major difficulty I encountered this summer involved switching from Swarm's random number generator to Java's. Swarm used the method *getDoubleWith$withMax()* and because I couldn't find a comparable one in Java that allowed me to specify the lower bound, I used a math formula to adjust for this. However, this resulted in a null pointer exception, so I switched to RePast's version of Swarm's method, *getNextDoubleFromTo()*. Since it is very similar to the Swarm method, implementing it was easy. The only difference was that the Swarm next double works on [min, max), while the RePast one works on (min, max). Had I found this method sooner, I would have implemented it before attempting to use Java's method of *nextDouble()*.

When I started working directly with the RePast toolkit, I naturally wanted to compile some of their source code and run the program. At first, I had trouble doing this – I got an exception in the main thread because its definition was not found. I talked with Xiaorong, and she told me this was happening because their source code all had a package statement at the beginning. To make it simpler for me, I removed the line from their source code and was able to compile their code by typing *javac *.java* and then executing it by typing *java SimulationName*. An alternative method to erasing the package lines would be to run the simulation from the specified directory.

As my GUI progressed, I added support for the graphs. RePast's website said that graphing could be very slow, and it was. However, when I ran the GUI on Xiaorong's much faster computer directly (I would normally ssh to it), it graphed without a noticeable delay. To correct the massive slowdown on most computers, I added a parameter that allows the user to specify how often he wants the graph to be updated.

NOTABLE METHODS ADDED

MODELSWARM.JAVA

printStats()
zeroStats()

MOLECULE.JAVA

draw(SimGraphics g)
getProbedProperties()

STARTMOLECULE.JAVA

begin()	buildSchedule()	setup()
buildDisplay()	plotPoints()	step()
buildLegend()	populateNumAdsorbedX()	

NOTABLE METHODS EDITED

MODELSWARM.JAVA

buildModel()	react()
move()	sorption()

MOLECULE.JAVA

move()	sorption()
react()	

STARTMOLECULE.JAVA

main(String [] args)

LINKS

RePast	http://repast.sourceforge.net
API Documentation	http://repast.sourceforge.net/docs/api/index.html
FAQ	http://repast.sourceforge.net/modules.php?op=modload&name=FAQ&file=index
How-To Documents.....	http://repast.sourceforge.net/modules.php?op=modload&name=Sections&file=index&req=listarticles&secid=2
Java	http://java.sun.com
API Documentation	http://java.sun.com/j2se/1.4.1/docs/api/
NOM Simulator	http://tobit.cse.nd.edu
Simulation Manager.....	http://tobit.cse.nd.edu:7777/nom/simmgr.jsp